Microsoft .NET Deck Delivery

Version 1.6
June 1, 2001

Using this document:

This document is a supplement to the Microsoft .NET deck, and is designed to serve as a guide to assist you in the delivery of that deck. It goes beyond the speaker notes in the deck, and includes overviews of each of the slides, a script to assist you with figuring out how to talk to each slide, and references for more information if you want to more deeply understand some topic.

Introduction

The introduction. You should probably fill in your own name and title on this slide prior to getting up in front of any customers.

Changing Environment

Key concepts:

- Role of IT is changing
  - Expectations are that IT will help drive revenue and profits, not just cut costs
  - IT needs to integrate with the outside world (suppliers, partners, customers)
- Traditional business requirements still apply
- .NET is about helping people build more successful, agile businesses in this environment

Script:
Let me give you a little bit of context on where we're coming from. As we start from a business perspective, there are a lot of new demands on companies, and a lot of these really relate to the changing role of IT.

Historically in a lot of ways IT has been a cost center, and IT has been focused on what happens inside the organization. But as we move into this digital economy, there are a lot of pressures on IT. Senior management is suddenly very concerned about how information technology can play an additional role in business. And they're looking to IT to help drive revenue growth, profit growth, and to help build a sustainable competitive advantage.

At the same time, the world that was nice and tightly constrained within the four walls of the organization – suddenly people are being asked to interact with partners, suppliers, and customers all over the world using the Internet as a connectivity medium. Employees are no longer being nicely behaved and sitting at their desks between the hours of 9 and 5, they're working all over the world. And so there are a bunch of these new pressures.

Simultaneously, there are some traditional demands from a business perspective. It turns out it's good to bring in more money than you spend. And one of the real challenges is: how do we balance these two sets of demands, in terms of the increased technology demands and traditional business requirements?

That, at the 50 thousand foot level, is where we start from a .NET perspective:
How do we help people build more successful businesses, and more agile businesses for the digital economy?

Technology Drivers

Key concepts:

- Continued increases in computing power combined with increases in connectivity give us powerful engines to use to build better solutions
- Internet-connected devices should be accounted for and treated so as to take advantage of their unique capabilities

- Simple, ubiquitous standards are driving change as well

Script:

At the same time that you have those business prerequisites, technology doesn't stand still. Moore's Law continues to move forward; processing power is abundant, and Moore's Law probably has at least 5-10 years left on it, which is probably the extent of the visibility people have. So we can expect to continue to see abundant processing that doubles every 18 months.

That's starting to have a very significant impact, both at the very high end and at the very low end. You're seeing that very high-volume low-cost PC technology move into the biggest iron applications. At the same time, my new cell phone has as much CPU power as my high-end PC did 5 years ago.

At the same time, there's a second very powerful economic engine, which is the plummeting cost of communications. Certainly on the backbone of the network, you can argue that there's actually a glut of bandwidth. And the cost improvement curve is actually probably even steeper than what we've seen with Moore's Law. We still have a long way to go to push that bandwidth out to the edge of the network, but that's happening slowly but surely.

One of the big questions is in this world where you have lots of processing power and lots of communications is, "What's the right way to architect your systems to take advantage of that?"

Microsoft and other technology companies have built our businesses around this idea that Moore's Law was like having the wind at our back. It's the thing that will allow us to make a set of investments this year and know that next year they're going to run more quickly and more cheaply. Well, we still have that wind at our back, but we also have that second very powerful engine which is the plummeting cost of communications. And what we want to do is build a platform that takes advantage of and harnesses both of those engines.

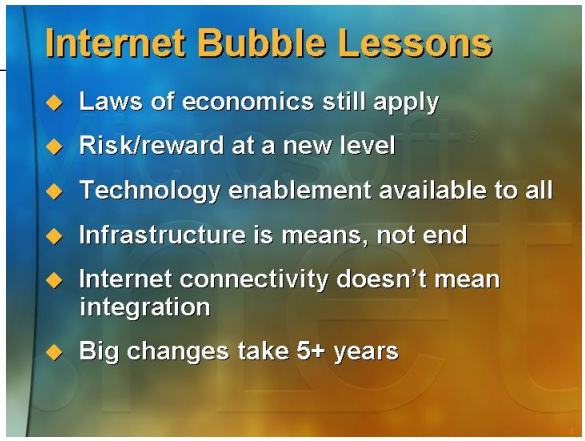
In a lot of ways, at kind of an economic determinism level, that's what .NET is all about.

There are actually some other interesting phenomenon happening from a technology perspective. Device proliferation – the Internet or the Web up until today has really been a PC-driven phenomenon, we're now seeing all kinds of new devices that are connected to the Internet. It turns out that a bunch of these aren't as interesting as we thought they'd be. There's a lot of work to be done to make the cell phone, for example, a much more interesting and smarter device that can do something interesting – but that

opportunity is there. And as we build architectures we have to think about how we distribute information and services across these different devices and do it in ways we're going to exploit those capabilities uniquely.

Finally, there are a bunch of interesting new Internet standards. Having a ubiquitous set of simple standards for connectivity around the Internet with TCP/IP, around the web with things like HTML/HTTP – those things had a huge impact. They were very broadly acceptable, they were things people could take advantage of without having to go out and hire a boatload of consultants, they were very organic and very decentralized.

There are a similar set of standards that are emerging in terms of how applications talk to applications. The Web has revolutionized the way users talk to apps, but it's done very little for how applications talk to applications. We live in this world with lots of connectivity, where in theory any application inside the firewall, outside the firewall – in theory those things can talk to each other. But we don't have a lot of constructs to get any kind of meaningful discussion between them. And so these are some of the drivers we think about.


Internet Bubble Lessons
Laws of economics still apply
Risk/reward at a new level
Technology enablement available to all
Infrastructure is means, not end
Internet connectivity doesn't mean integration
Big changes take 5+ years

Internet Bubble Lessons

Key concepts:

- There are a set of lessons from the Internet bubble that are relevant to all companies
- Prediction about five years ago was that the Internet was overhyped in the short run and underhyped in the long run. Most people described the short run as 5 years – we're now at that crossroads
- Laws of economics – want revenues to exceed costs
- Technology foundation we were building on during bubble wasn't adequate

Script:

Every few decades, the way we think about handling information and using computing changes fairly dramatically, from the centralized mainframe era with databases to the world of personal computing to the last decade, where there's been a lot of Internet publishing, a modest amount of transactions, but really just using the Internet to present screens.

We haven't taken the Internet and turned it into a data bus. We haven't taken it and made it so that you can see information from different sources

and that it fits directly into both your knowledge worker environment and your enterprise software. And the benefits that will come as we do that are pretty incredible.

The pieces that have to come together for this do take time. So we think this next decade is the big one. This is the decade where your involvement with computing will be very pervasive -- from reading to meetings, in every way.

Now, given that bullishness, I think it's interesting to talk about what happens -- you know, what about all these promises that weren't fulfilled? Is this just another case where somebody is being overly optimistic? Hasn't reality set in? Certainly one symptom of reality setting in is what's happening with the pricing of technology stocks. The NASDAQ last May was about 4300, off from the peak of 5000, but now we're less than half of that peak.

So some of the mania about this really changed. People have stepped back and even in some cases wondered, "Is there anything magical here at all? What is this all about?" In the extreme, of course we all have our favorite examples of Web Sites that we used to visit that are no longer there, and many of these companies, in retrospect you look back and say, "What was that all about? Why was that IPO such a big IPO?"

People talk a lot about the new economy -- or they did talk a lot about the new economy. It turns out that the rules of profitability or cycles in the economy have not been suspended. So a lot of what we're trying to do is to help businesses use the Internet to address real business needs.

So the lessons are pretty straightforward. That equation about profitability still applies. Technology enablement is not something that just startups have available. That was always one of the great paradoxes of the Internet. How could a technology that reduces barriers to entry allow the creation of companies whose value was greater than the ones who created assets in the real world where you build stores and there are real barriers to entry? You know, that paradox has now been resolved. There was no reason for that to take place. But for a while it was interesting.

You know, the signals the market was sending were somewhat confusing. There are people who actually spun off their dot-com operations to monetize that, which is a very rational response to what the market was doing. Virtually all of those now have been spun back in.

So all this technology enablement, it's great for the companies and in a way it allows you to magnify your excellence. You know, if you make the best product in some category, now your cost of getting customer feedback and

distribution actually can be significantly reduced. If you were the number two in a market, the fact that the number one guy can get those distribution capabilities could be tough.

And a final point here is just that things take time. You know, if you take things like credit cards that today we take for granted, look at how long it took for people to just take that as common sense. Well, buying on the Internet is the same way. I have no doubt that Internet retail sales, consumer retail sales over the next five years will grow by about a factor of four or five. But they'll grow in a very linear way as the word of mouth, as the ease of use, as just the common sense that, "Hey, this is a good way to do things; it actually works," suffuses throughout the population and different age groups.

More information:

Integration Imperative

Key concepts:

- The integration problem is the key need we're working to address
- Solutions increasingly involve more than one application
- Integration right now is hard and expensive

Script:

One of the biggest catalysts for change in how we create software solutions, particularly in terms of the corporate perspective, is this integration problem. We have this sea of connectivity with all of these islands. We have islands of data, we have islands of applications, we have islands of organizations. How do we start to get those things to talk to each other on our behalf?

What's happening is that we're seeing what we think of as a solution evolving from being a single standalone application to increasingly being multiple apps, multiple devices, multiple services that are harnessed together into constellation, and we want those things to march together in the same direction, working together to do something on our behalf.

That historically has been a very hard problem. With integration today, we build applications, then about ten minutes after we're done with them, it dawns on us that we actually want to integrate them with a bunch of other

things. So we get out the duct tape and start wrapping it around the application – or more likely you go out and buy some very expensive EAI software.

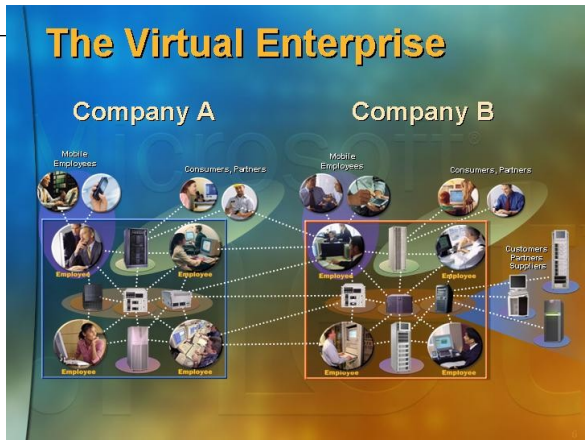
You can spend millions of dollars on EAI stuff, for every dollar you spend on software you end up spending $7 to $10 on the services. Really, custom, ad-hoc, one-off code: glue code. And as soon as you change any of the endpoints in that constellation, you tear that glue code away and you start over again. There's no reuse.

The opportunity for improvement is because this stuff is so hard and so expensive and takes so long, and once you actually do the integration you actually lose an enormous amount of flexibility. Once I've got two applications hardwired, that turns out to be a real barrier to saying "You know what, I don't like this application anymore" or "I don't like this partner anymore. I want to go plug into this other thing." It becomes a huge barrier when you've done all of this work. You have huge switching costs when these things are hardwired together.

So what we want to do is make that much, much easier.

This integration problem pops up again and again and again. Anything you're doing inside the organization at a business process level will span multiple applications, and often span multiple organizations. How do you streamline this?

The Virtual Enterprise



Key concepts:

- Enterprises need to worry about integration
  - Between applications inside the corporation
  - In their supply chain
  - With customers
  - For their employees
- It's no longer possible for IT to address only what happens inside the corporate firewall

Script:

If you look at any of the B2B scenarios – whether that's with partners, suppliers, or customers – they're all about how we get computers to talk to other computers, except in this case they're in different organizations so they need to traverse firewalls, they need to deal with security issues, they

need to deal with Internet reliability issues. The traditional mechanisms for doing that kind of integration are too expensive, too hard – but also don't work across the firewalls. The last thing you want to do is use one integration methodology inside the organization and then have something else outside the organization. You end up being forced to make this investment in bridging, and you'll end up making this investment in two parallel sets of skills. And that's just very difficult.

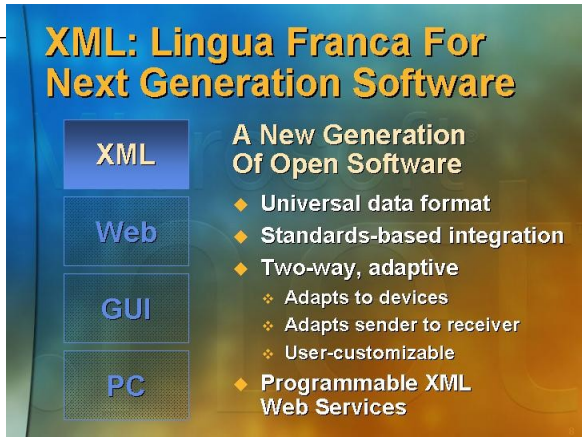
It used to be that you only had to worry about what was happening inside your four walls. You could control what applications were deployed and you could, where necessary, hardwire the integration you needed between applications. Now, though, there's a need to project this information outside of your four walls, to your partners, you customers, your suppliers. So you have to worry about security. You have to figure out some way to give your mobile employees, who want access 24 hours a day to the information inside your corporation, the ability to get at and act on the information they need, without compromising your systems. So the nature of enterprise computing is undergoing this big transformation, from an environment in which you could be very insular to one in which you need to reach outside the firewall to all kinds of people and systems in a secure way.

The integration problem pops up in a bunch of other places, too. You start to think about the technology in your life. There are a variety of applications, a variety of devices, a variety of services. Those things don't work together particularly well. Today you have to adapt to each of those different things. You have different passwords, different usernames for each different web site that you might interact with. Getting data from one device to another device is really difficult. Try to get a phone number from your cell phone to your contacts database on your PC. That's a nontrivial exercise today. So you end up being the integration point for a lot of these different things. And there's an opportunity, because we live in this world of ubiquitous connectivity, there's an opportunity to get these computers to talk to each other and do stuff on our behalf. And that really at its core is the problem we're trying to solve with .NET.

XML: Lingua Franca

Key concepts:

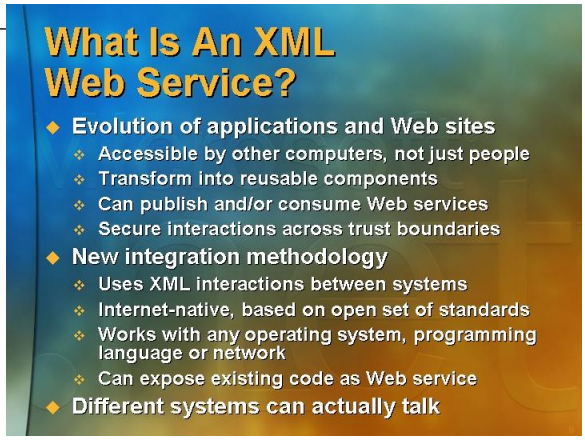
- XML is a simple technology which allows programs to talk to each other meaningfully
- The impact of XML will probably be as big or bigger than the impact PCs had



Script:
So, as we think about solving this integration problem, the technology that we're relying on very very heavily is XML. XML is a deceptively simple technology, but we think that it has the potential to have an even bigger impact than some other very big enablers – whether that's the PC or the Web itself.

XML is about expressing data in a way that computers can understand. It's a standard, it can be implemented anywhere, it's a very simple technology. Once we get the data into XML, we can transform it, we can translate it, we can slice it, we can dice it, we can get it to wherever we want to get it. And computers can be smart about figuring out that information so that we start to have the prospect of systems that were designed by different people having meaningful interactions. That historically has been extraordinarily difficult because everybody's been building their own islands, and they haven't thought about "How do I plug into the much broader environment that is out there?" And really feeding this is this idea of programmable XML Web services.

What is an XML Web Service?


What Is An XML Web Service?
Evolution of applications and Web sites
Accessible by other computers, not just people
Transform into reusable components
Can publish and/or consume Web services
Secure interactions across trust boundaries
New integration methodology
Uses XML interactions between systems
Internet-native, based on open set of standards
Works with any operating system, programming language or network
Can expose existing code as Web service
Different systems can actually talk

Key concepts:

- The Web has been about user-driven access to information; XML Web services are an evolution to programmatic access to information and services
- XML Web services are like components
- XML Web services are about making it possible for software to talk to other software without requiring a human in the middle

Script:
Up until now, the Web has been about interactive user-driven access. You as the user click on a link, you type in a URL, you go visit a website. And you can sit down, type in URLs and there are tens of millions of websites out there. But if we want any two of those websites (which are basically applications) to talk to each other, the Internet doesn't do a lot for us.

So what we want to do is start to think about how computers can talk to other computers, so it's not just users that are accessing that application, but other computers have the ability to do that as well.

The best analogy for this is what happened with component software. Component software works because we had a set of conventions to package up an arbitrary chunk of intellectual property or code. And once we package that up, you just start to treat that as a building block. You can grab a component from over here, a component from up here, a component from over there, and you can start to stitch those things together. From an economic perspective, it had huge value for developers. You didn't have to do everything yourself. You could start to make make-or-buy decisions. Do I want to write a charting module, or do I want to take advantage of one off the shelf? Well, what's the price? Maybe charting's really really important to what I'm trying to do, so I'm going to do that one myself. But you start to have a lot more choice; there's a lot more granularity in the systems.

And if you look at something like the COM market today, it's probably on the order of a billion dollars a year, third-party components on the Windows platform. And that's a huge economic phenomenon because it just gives developers more choice in terms of what they have to build and what they don't have to build. Nobody has enough time, nobody has enough resources to write all the code they want.

So the premise of XML Web services is to do the same for any app that is Internet-connected. So any application, any device, any service starts to become a reusable component. And I can grab a service from over here, a service from over here, a service from over here, and I can start to integrate or aggregate those things together to build a bigger solution.

So you want to construct your capabilities as a set of these services. You have the ability to expose them as a web service so that the people you want can take advantage of them, but you also can start to make these make-or-buy decisions to consume them as well, taking capabilities that are inside the organization and using capabilities that you can buy off the shelf, services that are out there in the cloud. You can also start to bring in something from your partners, and that's the way we start to hook up with those arbitrary partner applications.

So XML Web services is really this idea of an Internet-native integration methodology: how do we get systems to start to have meaningful discussions with one another?

And I'll walk you through this stack in a second, but basically it's an open set of standards and it can be implemented on any kind of system. So you can take existing systems and wrap those and expose them as an XML Web

service. You don't have to throw everything away and start over again. (We've found that that's not a great proposition to go to people and say, "Hey, you should junk all of your stuff.")

And the end result of this is that you can start to get different systems designed by different people to start to have a meaningful discussion. Historically, that's been a very very difficult thing to do.

XML Web Services Foundation

Key concepts:

- Simple standards stack
  - Internet – lowest cost way to move bits, broadest reach, deepest investment
  - XML – W3C standard
  - SOAP – W3C effort
  - UDDI – open, founded by MS, IBM, Ariba, over 200 members
- Easy integration, distributed control
- Vendor, platform, language neutral

Script:

So the stack for XML Web services is actually a very simple stack. Using the Internet for ubiquitous communications seems like a pretty safe bet; very broad reach, there's a huge amount of investment there, and it's the lowest cost communications fabric the world has ever seen. Above that, we're going to describe our data in XML. Once we get our data into XML, we can slice it, dice it, manipulate it, and transform it into whatever we need. And it can carry a bunch of metadata along with that, so it's much easier for computers to figure out what other computers are talking about.

But we need to do more than push data around. We also need a way for commands to be passed back and forth between systems. Today, you do this with a tightly-coupled remote procedure call. And that model works pretty well if those systems are very very localized. But when you start thinking about doing RPCs over the Internet, across firewalls, across non-reliable network connections, you start to have a lot of difficulty doing this. And SOAP is a loosely-couple, message-oriented mechanism for sending commands back and forth. It's very Internet-native, it's all XML, it can go anywhere the Internet can go. It doesn't punch a big hole in your firewall in order to get a binary protocol through those things. And if you think about the really big, really successful systems today, the interactions between different components are actually loosely-coupled and message-oriented.

It's bad that it took us 10-15 years to figure that out, but we've finally figured that out.

So you're still going to do tightly-coupled interactions in the small. Locally. But when you start thinking about the interactions with systems that are nowhere close to you, you're going to use this loosely-coupled message-oriented model.

Now assuming we're successful with this, you're going to have hundreds of millions of services out there that are accessible to people programmatically. You need a way to find those things, you need a way to describe those things. UDDI conceptually lets us describe available Web services. It also gives us a directory structure kind of like DNS for Web services so that we can build directories, we can build search engines so that you can go out and find an arbitrary service. "I'm looking for a service that does X, Y, and Z."

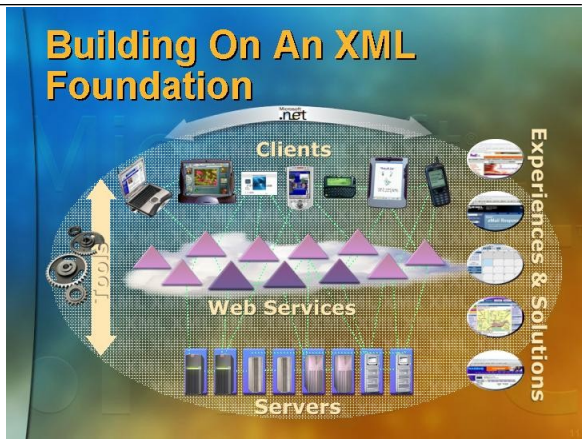
There are a couple of things that are interesting about this stack. One is it's very simple. So like HTML and HTTP, it's pretty easy for a lot of people to get their hands around it. It's not serious rocket science. And that means that as we start to build up this world organically, it's easy for lots of people to participate.

Secondly, these are an open set of standards. Microsoft has been very involved in writing them, whether it's been XML or SOAP or UDDI. We could have done those things ourselves, but instead we've gone and worked with the W3C and with a bunch of other companies to make those things very very broadly acceptable – because what we want to do with .NET is provide a platform that lets you orchestrate and integrate lots and lots of different endpoints. Not just Windows endpoints, but any kind of endpoints.

Now it's important to be more than open. "Open" is a fairly abused term. (We joke from a marketing perspective that it's roughly a four character expenditure.) What's really important about this model is not just that the specifications are there and anybody can take advantage of it, but we actually have seen extraordinarily broad adoption across the industry. I'm hard-pressed to think of a vendor in the industry today who isn't committed to supporting this model of loosely-coupled XML-based interactions between systems using SOAP and UDDI. I'd be very surprised if there was anybody who isn't on board with this.

So we've done a really good job of getting the industry buy-in to this whole integration methodology.

Building on an XML Foundation



Key concepts:

- There are five key areas of deliverables to create an XML Web services platform
  - Clients
  - Services
  - Servers
  - Developer tools
  - Experiences and solutions
- .NET is Microsoft's software platform which addresses all of these areas

Script:

So, assuming we're going to use the Internet as this big XML message bus, where we've got data and messages flying all over the place, there are a number of different things that happen in this world.

And we start to think about clients that know how to consume these services and do interesting things with XML, we think about a set of server infrastructure that knows how to serve that stuff up to different clients and different services, and we think about the new components that are the XML Web services, the idea of resources that are accessible via the Internet using this XML methodology that allow us to grab an arbitrary package of functionality. So most of the interactions in this world are clients, services, and servers interacting back and forth.

There are other pieces. There are tools, and you'll hear a bunch about our various tools, but we've made a huge investment in this methodology. We started in 1996 to go back and rebuild all of our lowest level developer bits around this methodology. It's taken us a lot longer than we would have liked to get that stuff done, but we'll ship Visual Studio.NET that lets you write code that just intrinsically sits on top of this XML foundation, and we'll ship that stuff finally later this year. But we're providing a toolset that works at the client level, at the services level, at the server level. How do I start to aggregate a service from over here, a service from over here, and a service from over here? How do I start to think about business processes that span a whole bunch of these different nodes? There's a very significant investment in terms of tools for developer infrastructure.

On the far side, you'll see experiences and solutions. The premise is that the new solution is not just a standalone node, but it's a whole bunch of nodes working together. Things that come together in a constellation of multiple apps, multiple services, multiple back end servers, multiple

devices, whatever it may be. And I'll give you some examples of some of the things that Microsoft is building in this space, but we think that that increasingly is the level that people who are building solutions are going to operate on.

Benefits of XML Foundation

Benefits Of XML Foundation
User: Beyond browsing
Access and act on information
Greater control, better decision-making, easier collaboration
Optimal support for different devices
Business: Integration by design
Best-of-breed internally and externally
Business processes and partners
Agility to run, adapt business better
Industry: New opportunities
Don't have to do it all yourself
Make your value easier to consume
"Plug and play" business relationships

Key concepts:

- More efficient and flexible business processes
- User Experience at the fore, technology in background
- Don't have to do it all yourself – XML Web Services can be consumed

Script:

There are a bunch of benefits of this XML foundation. From a user perspective, we can start to deliver a much richer experience. The browser's been a huge phenomenon, but it's fundamentally dumb terminal computing. You do everything on the server side, and periodically you render up a picture of what happened on the server. If you're doing a bunch of compute-intensive stuff – particularly in a world in which there's lots and lots of processing power out on the edge of the network – that model doesn't make a lot of sense. If you're doing, for example, engineering design, the last thing you want to do is have to do a server roundtrip every time you want to recalc a particular model.

The Web's been successful because it's provided a very simple application model but it still had a lot of the same problems we saw with the mainframe model. We moved beyond the mainframe model because processing power was abundant: we could build a richer experience. With .NET, we'll really push you to deliver the software you need to build, much richer software that distributes load between smart clients and smart servers. From a user perspective, that means you get a much richer experience. It's not just about browsing, it's about being able to actually do something with the information.

We're going to push that data out in XML. That means that you're not screen-scraping an HTML picture, but you can actually get that data and do whatever you want with it. You can manipulate it, you can analyze it, you can annotate it, you can send it back into the system. From a user perspective, that's going to give people a lot more control and flexibility, that's going to give people the ability to make better decisions because they're going to have more data and they're going to be able to do more

with it, and it's going to be much easier to do collaboration. You're not just going to be talking to that single island, but you're going to have a better process for being able to talk across different systems.

And the PC will continue to be the smartest device in its food chain, in its information ecosystem, but we have lots and lots of other smart devices. And people are going to go write a lot of code to make those devices more interesting. You'll hear people talk about thin client vs. thick client, or 6-12 months ago there was a battle about PCs vs. non-PCs: we think the interesting dynamic is smart clients vs. dumb clients. Smart clients are more interesting because they have more software; they're going to be more valuable. The PC obviously is the smartest of those clients, and you're really seeing this play out with the new classes of devices that are out there. Whether it's Palm or Pocket PC, those devices are successful because they're smart about synchronizing with the information you care about on your PC. If your Palm or Pocket PC was a standalone island of data, it would be a lot less interesting. The RIM Blackberry is successful because it's smart about getting to your email.

Contrast that with dumb devices like the network computer that the mainframe guys keep flogging and flogging as the new 3270 for the 21st century. That device hasn't been particularly successful, and the reason is that it's fundamentally a dumb terminal. You can dress it up, but you can't take the "dumb" out of the dumb terminal.

The other interesting example is what's happened with the WAP phone. A year ago there was a lot excitement about what the WAP phone was going to do. And I don't know if you've played with these but what's interesting to me is that it's not only a dumb terminal, it's a really dumb dumb terminal. In order to make that a really interesting experience, we're going to have to add a tremendous amount of smarts to overcome the limitations in screen size, the limitations in network bandwidth and latency, the limitations in the input model. (Do you like typing on this little keypad? Quadruple-typing to get a character, that's fun.)

So there's a huge amount of work to be done, but we think you're going to start to see more and more software across all these different devices. And the key to really making that world work is to have an application model that allows smart devices to talk to smart devices and not say that we have to centralize everything on the mainframe. So we think that'll be great from a user perspective, and from a business perspective its just going to make integration so much cheaper and faster and easier than it is today. At the systems level, at the partner level, you're going to be able to do plug-and-play integration. I'm going to take this system and this system, I'm going to take this partner and this partner. So we're going to be able to integrate different systems and have meaningful dialogue between them at a much

easier faster quicker pace than we've been able to do it historically. And not just get that integration done, but have the flexibility to adapt and evolve as business conditions change, and as partners change.

Finally from an industry perspective, you don't have to do everything yourself. One of the lessons people learned in the dot-com era is that it's very hard to build everything yourself. The idea that you were going to put up your own island and users were going to flock to it – it turns out that that requires an awful lot of investment. What you really want to do is create an asset that can work well with the other things in the customer's environment, whether that's at the individual level or at the organizational level. As we start to move away from this idea that everything is an island, that you can actually deliver functionality that can play well with others, that creates a lot of opportunity across the industry. You can take your functionality and expose it in a way that's much easier for other people to consume. Similarly, if you're making make-or-buy decisions for development, there are more capabilities that you can take advantage of off the shelf.

Microsoft .NET
A Software Platform for XML Web Services
Spans clients, servers and services
Consistent programming model, highest productivity tools for deepest developer pool
Interoperable via XML Web Services
Key capabilities
Deliver software as a service
Integration becomes intrinsic
Enable smart, service-aware systems
New, richer user experiences
Microsoft .net

Microsoft .NET

Key concepts:

- .NET is our software platform for the XML Web services environment
- Continuous delivery of value is the goal of software services
- Software services also make it possible to tailor software to companies and individual users
- .NET will make building, deploying, operating, and aggregating XML Web services easy

Script:

So as we think about .NET, .NET is our software platform for this XML environment. And we're building client software for this environment, server software, a newest of services that I'll talk about in a few minutes, we're also making a huge investment from a developer perspective with a new set of tools, and a whole bunch of infrastructure and plumbing that will make it very very easy to build these things. XML is great, but you don't actually want to be parsing a lot of XML. We're providing higher-level frameworks and tools that allow you to tee up a set of interactions between these things; we think this is very very important.

There are a couple of key tenets that you should think about around .NET. One is that we're evolving to a world of software services. So instead of Microsoft and everybody else every couple of years we do a release, we put it on a CD, we throw it over the wall, and hey we're done. We're going to move to a world, because we have this ubiquitous connectivity, to a world where there's a much more continuous relationship with that end customer. You're going to see a continuous delivery of value. We're going to push updates, we're going to push bug fixes, we're going to be able to do a much better job of tailoring our software for different audiences.

You can imagine an infinite number of configurations of your software. You look at something like Office today, this is a very large piece of software which has very few configurations. There are a bunch of economic factors that conspire to make us say, "There will only be three boxes." The test matrix is such a mess. It used to be that retail shelf space was a big issue. So when we start to think of these things as services, you're freed from some of those constraints. You can start to do a much better job of tailoring, literally down to individual users.

Another key tenet of .NET is obviously making it really easy to build and deploy and operate and aggregate and integrate and consume these different XML Web services. By nature all of the software we're building is going to speak XML. And that means that integration becomes intrinsic. It's not an afterthought like it is today. So we're going beyond the dumb terminal to build great smart devices that are not only rich in functionality, but are also easy to manage and deploy. That's kind of been the fundamental tradeoff over the last five or six years, you could build an application that was easy to deploy, or one that was rich in functionality, but doing both was extraordinarily difficult. And to a large degree Microsoft is responsible for that tradeoff. "DLL Hell" wasn't invented by design, but it was a characteristic of that ecosystem that really enforced that tradeoff. Fortunately, since we created that problem we can fix that problem. That's one of the things that the .NET Framework and the new set of tools offers: the ability to build rich functionality that can also be seamlessly deployed.

Finally, building on top of all of these capabilities, we're going to be able to build new, much richer user experiences. We're going to be able to supply code to do more cool things for our customers, and go beyond the limitations of a browser or a more traditional standalone application.

So let me go through some of the software deliverables we'll be building. I mentioned that there are clients, there are servers, there are services, there are tools – we think that there will be a lot of different people who are going to play in this world of XML Web services at different levels, and Microsoft is making a pretty systematic across-the-board investment.

.NET Clients

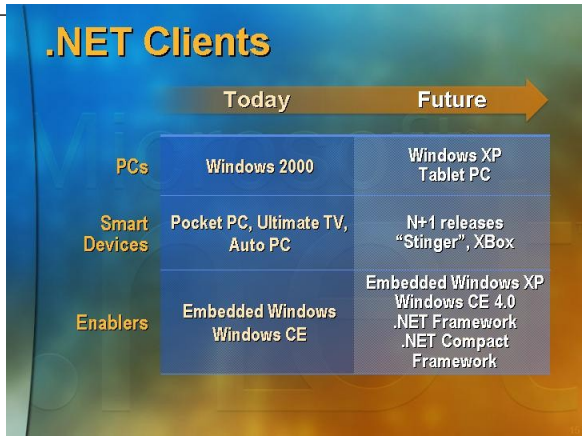
Key concepts:

- Smart devices are
  - Smart about the network (online/offline)
  - Smart about other devices
  - Smart about services
  - Smart about you and your information
- Microsoft is building XML into all of our client software

Script:

On the client side, we're obviously investing in a broad range of clients. Certainly, from a PC perspective, we've already made significant investments to make the PC a great XML client; Windows 2000 was the first operating system with native XML support and we've continued to build on top of that over time. But we're also building a bunch of these other smart devices, whether those are mobile devices, or new form factors like the Tablet PC. So we'll use Windows as the core smarts to power those devices so that they're service-aware, they're XML-aware, they provide a great standalone experience but also designed to work as part of these larger constellations, so that they're not just an island but also play well with the other devices you care about.

Clients Roadmap



Key concepts:

- Windows 2000 was the first OS with native XML support
- Windows XP extends on that with native SOAP support
- N+1 releases continue to build XML and SOAP deeply into clients

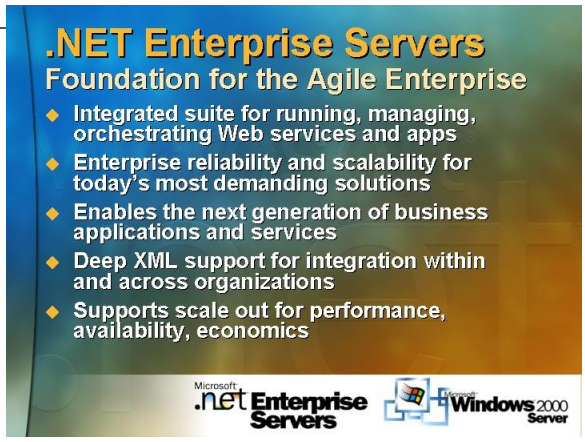
Script:

In terms of roadmap, Windows 2000 is obviously there today. Windows XP is going to be a big release from a consumer perspective, because we're going to take that Windows 2000 codebase into the consumer space, there's

a bunch of additional XML functionality there, SOAP support is there, and you'll also see things like Tablet PC which use the same codebase and continue to push this idea of smart devices – *really* smart devices – in other form factors.

Things like Pocket PC, Ultimate TV, etc., those things are out today, and in the n+1 releases of these you can expect that deep XML support to show up, as well as the initial releases of the Stinger cell phone and Xbox. There are also a lot of enablers for people building appliances or other form factors that include various embedded Windows operating systems, and also some of the core development functionality for this next generation of apps: the .NET Framework and the .NET Compact Framework which is a very small version of that with a footprint that's very small so that it will fit on teeny little devices.

.NET Enterprise Servers


.NET Enterprise Servers
Foundation for the Agile Enterprise
Integrated suite for running, managing, orchestrating Web services and apps
Enterprise reliability and scalability for today's most demanding solutions
Enables the next generation of business applications and services
Deep XML support for integration within and across organizations
Supports scale out for performance, availability, economics
Microsoft .net Enterprise Servers
Microsoft Windows 2000 Server

Key concepts:

- We're providing a broad, integrated server family
- Design tenets:
  - Integrated to work well together
  - Support for XML
  - Support for scale out
- Store, route, transform, bridge to legacy data

Script:

On the server side, we have a very comprehensive family of servers where the central defining point is building a suite that provides the infrastructure for helping people build, deploy, operate, and orchestrate with these new XML-based solutions.

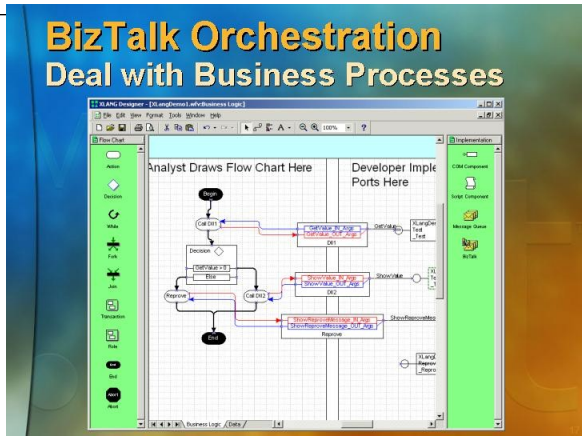
We have a whole bunch of products, whether that's SQL Server, BizTalk Server, etc that do interesting things by themselves, but there are a couple of common design points across them. One is native XML support. So SQL Server 2000 is a native XML store, for example. BizTalk Server gives us a way to route and transform and start to describe higher level business processes in XML. Things like Host Integration Server, while they're not exciting, they do give us a way to reach out to that non-XML data world that happens to be the vast preponderance of the world today, and get that data into the XML world. So, with this wave of servers that have shipped over the course of the last six months or so, we've got the first wave of XML

functionality across these different products. You'll see us continue to deepen that.

Really, the core design principle across these things is that we provide ever deeper support for this XML world, you'll see us continue to raise this idea of software scale out. So instead of trying to run our applications on one very big box, we want to be able to scale by adding multiple boxes and get great performance, great economics, and great availability.

Windows 2000 makes a great foundation for this.

BizTalk Orchestration



Key concepts:

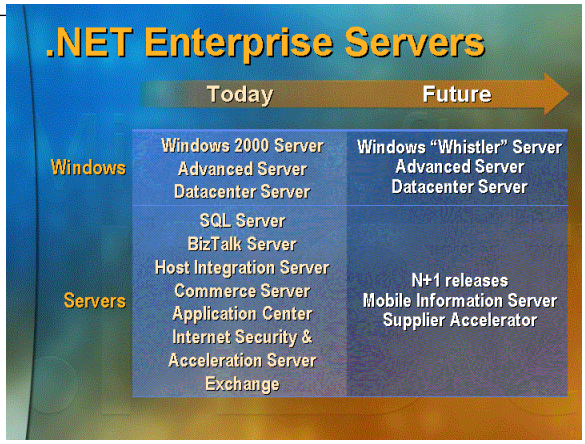
- BizTalk Server gives us an XML-based way to make integration (and business process orchestration) easier, cheaper, and faster

Script:

So here's a screenshot from one of the .NET Enterprise Servers, which is BizTalk Server. This illustrates how BizTalk makes integration faster, cheaper, and easier. It looks a lot like Visio because it's built on top of Visio, which is, it turns out, one of the most popular business integration tools today: draw a diagram, print the diagram, hand it to a developer.

BizTalk lets us separate the business process from the underlying implementation. It makes it easier to initially describe the business process initially, and to update and adapt it as the underlying systems and processes evolve over time. It's all based on XML – incoming data is mapped to XML, and the XML is mapped to the appropriate format for outgoing data.

.NET Enterprise Servers Roadmap



Key concepts:

- The initial server infrastructure has already been delivered
- These servers support XML and software scale out
- N+1 versions will continue to embed XML and SOAP support

Script:

.NET Building Block Services

.NET Building Block Services
Microsoft Passport
Microsoft Passport
Authentication service offers single sign-on capability for any Web site
Over 160 million active accounts
"HailStorm"
Set of user-centric, XML Web services to manage, protect personal information
Puts users in control of their "personal networks" for more consistent experience
New way for businesses to reach customers, with customers' permission
Open Access model allows any application, device or service to use

Key concepts:

- Providing core services for developers to leverage is nothing new; printer drivers or GUI toolkits are equivalent
- Building block services are globally accessible core services for developers to integrate
- Passport
    - 160m+ active accounts
    - growing by over 10m a month
    - Over 1.5b authentications a month
    - One of the ten largest web sites in the world, yet no one ever visits it
- HailStorm is a code-name

Script:

Clients and server software you've seen from Microsoft before. There's a new category of stuff where we start to think about building block services, and the best example of this is Microsoft Passport. Passport is actually one of the ten most widely-visited Web sites in the world. Except what's weird about that is that nobody actually goes there. It's extraordinarily rare for somebody to type in www.passport.com. What happens is that it provides a way to outsource your authentication process. When you go visit MSN, you go visit Hotmail, you go visit a couple hundred third-party sites, and you go to log on, that authentication process is actually outsourced to these other servers.

And it really opened our eyes to this idea of providing a set of building blocks that are available anywhere across the Internet. So historically our platform business has been about identifying things that are needed by the developer community and providing them in a very high volume low cost way. So the printer drivers in Windows: you can build your own printer drivers, or you can take advantage of ours off the shelf. The same thing with memory management, GUI toolkit, etc., but we always try to find things that lots of developers need that are very core modules, so that developers can say, "Do I want to do my own GUI toolkit, or do I want to take advantage of this off the shelf?" "Do I want to write my own printer drivers? Printer drivers are really *fun*, maybe I do want to do that one." Um, probably not.

Historically, we've always provided those building blocks that have run in the context of a single machine. Now we're starting to think about what the

set of services are that we can provide to developers that may be globally accessible.

.NET Building Block Services Roadmap

Key concepts:

- HailStorm builds on top of Passport
- HailStorm will be in broad beta in CY 2001
- HailStorm will be delivered early in CY 2002

Script:

So Passport's been pretty successful. In March we announced another set of services called "HailStorm". And HailStorm is a set of user-centric Web services. The idea is that they help you manage a variety of different types of personal information and they figure out how to deliver those things, manage those things, protect those things and securely share those things with the different technologies in your life, other people's technologies, other people, other entities. The basic premise is that instead of building stuff that's wedded to a particular device or a particular network or something else, think of these as a big set of services in the cloud that you can just pour into the different applications in your life, the different devices, etc. The idea is that you can use the HailStorm services to make all of the different technologies in your life work together, on your behalf, under your control. So you don't have to adapt to everything, you don't have to be the integration point between those different technologies.

HailStorm is interesting in that it's not really a business-focused thing, it's not really a consumer-focused thing, it's an individual-focused thing. But the idea is that it's user-centric, it's identity-centric, which is a concept that has a bunch of broader applications. In the case of HailStorm the ideneity that we're keying off of is you as an individual. It may span both your business and consumer contexts. You could think of other identities that could be used for other services, such as groups and organizations.

We're very hardcore with HailStorm that the user is in control. None of your information gets shared without your explicit permission.

Another one of the interesting things about HailStorm is that you don't interact with HailStorm directly, you generally interact with HailStorm via some other endpoint: some application, some device, or some service. Generally, you don't interact directly with the myCalendar service, you might instead interact with it through Outlook. Or I could go to the

Mariner's website and say, "Well, should I copy down their whole calendar and then go type it into my calendar? Or what happens if I can give them permission to write data to my calendar?" So I could go to the Seattle Mariner's site, and I could say, "Here's who I am, I'm going to give you permission to write the following data to my calendar."

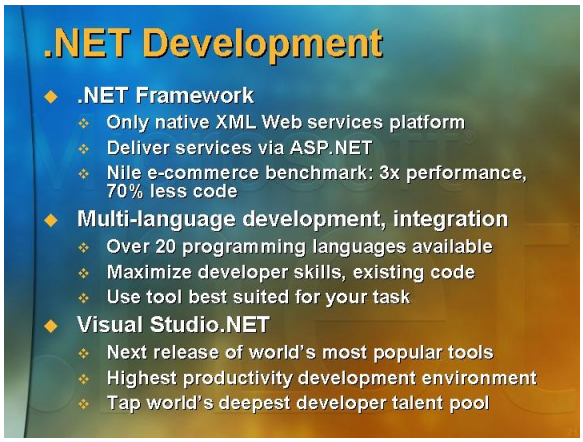
We use Passport as the mechanism to uniquely and securely identify people.

HailStorm is a huge step forward for privacy. Right now, your data is out there, it's on a bunch of islands, you have no prospect of getting it back, and no ability to control what people are going to do with it once you've given it to them. Really, the core value proposition of HailStorm is that this is a way to manage your stuff and your interactions with other people and businesses. Today, if you bought books at Amazon and buy.com and Barnes & Noble, one, you've got three different islands of data about your transactions that you can't control, and two, that data's out there and they can do whatever they want with it. In the HailStorm model, your stuff is your stuff, and it only gets out there if you explicitly share it, and you can't share any information without whoever you're going to share it with telling you "Here's what our privacy policy is, here's what we're going to do with your data." And the nature of the interaction between the systems actually lets us have a much higher level of recourse for end users if a site doesn't do what they say they're going to do in their privacy policy.

Now, a bunch of that data resides on Microsoft servers, and we've articulated a very strong set of principles in terms of what we'll do with that data. We're not going to mine it, we're not going to publish it, we're not going to target it, we're not going to sell it, we're basically not going to look at it. And today that's a set of principles, but there will be a set of activities that are embodied in licenses. But if we don't live up to that value proposition for protecting user data, then we're not going to be successful in this business.

We'll be in broad public beta on HailStorm before the end of this calendar year, and we'll deliver the services early next year. These are, of course, XML Web services, so the documents flying around are XML, and we've demonstrated clients on not only Windows but also Palm, Solaris, Linux, and a bunch of other systems accessing HailStorm data; we think that these services will be more valuable for users the more endpoints they can touch.

.NET Development


.NET Development
.NET Framework
Only native XML Web services platform
Deliver services via ASP.NET
Nile e-commerce benchmark: 3x performance, 70% less code
Multi-language development, integration
Over 20 programming languages available
Maximize developer skills, existing code
Use tool best suited for your task
Visual Studio.NET
Next release of world's most popular tools
Highest productivity development environment
Tap world's deepest developer talent pool

Key concepts:

- Microsoft is providing best of breed development tools with Visual Studio.NET and the .NET Framework
- The .NET development model embraces many programming languages, so that developers can use whichever language is most appropriate for them for that project

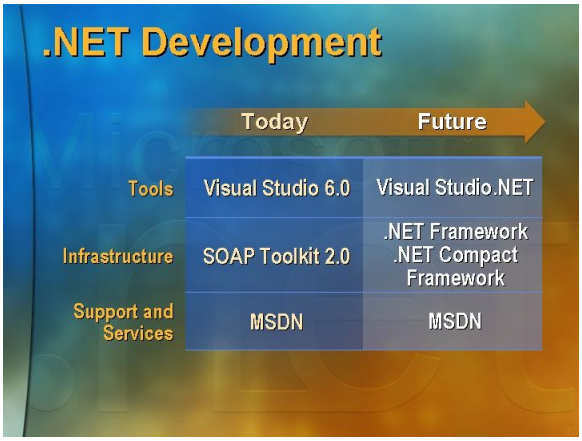
Script:

So if we want developers to build lots of XML Web services, it helps to provide them with appropriate tools. Microsoft is releasing a new version of the Visual Studio development environment – Visual Studio.NET – which is optimized to make the creation of XML Web services as simple as possible. Developers just create their applications in the normal way, but they then have the option of exposing their methods through SOAP, and using WSDL to describe their Web services. Access to UDDI is built in as well.

The .NET Framework is a product that forms the development foundation for the Microsoft .NET platform. The .NET Framework and the device-focused .NET Compact Framework provide a managed, secure execution environment for XML Web Services and applications, with comprehensive support for XML.

What are the benefits of the .NET Framework? Put simply, faster time to market, easier deployment and administration, and improved performance. The .NET Framework enables developers to use any programming language, and for applications written in any programming languages to integrate deeply with each other, enabling current development skills to go forward without retraining. It uses a highly componentized, plumbing-free design that enables developers to focus on writing business logic. Developers don't need to write IDL or Registry code, and ASP.NET, for example, includes dozens of controls that encapsulate common programmer tasks such as a shopping cart. The .NET Framework also includes technologies to make applications more reliable. For example, memory, threads, and processes are managed by the .NET Framework to ensure that memory leaks don't occur. And ASP.NET monitors running Web applications and can automatically restart them at administrator-defined intervals.

.NET Development Roadmap

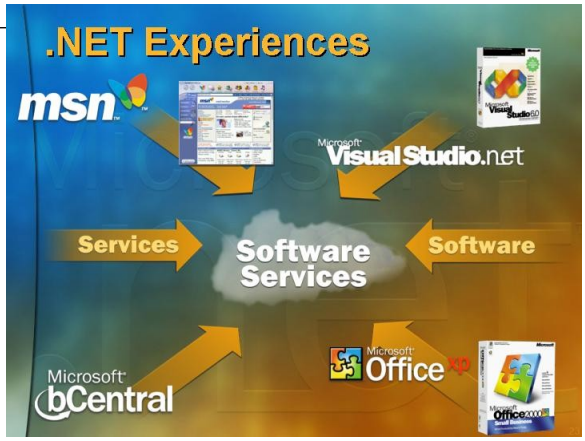


Key concepts:

- Developers can build XML Web services now with Visual Studio 6 and the SOAP Toolkit
- All of the significant development tools necessary will be delivered this year

Script:

.NET Experiences



Key concepts:

- .NET experiences combine the best of local applications and Internet services

Script:

Given all these nodes out there that are XML connected, you start to pull together solutions. Conceptually you can think of these as building a solution that is going to be a constellation of multiple applications, multiple services, multiple devices. We think all of these things are going to be software services. And what you're seeing is that more software capability is being added to the services, ad more services capability is being added to the software.

Looking at Microsoft's big customer-facing properties, whether that's MSN from a consumer perspective, bCentral from a business perspective, Office from a knowledge worker perspective, or Visual Studio from a developer perspective, what you really have are two services that are adding a bunch of software content. So things like MSN Explorer add code as software to make that a more interesting services experience. Similarly, Visual Studio is a standalone application or development environment, Office is a standalone application – those things are adding services capabilities. And it'll happen in an incremental way, but what we're moving towards is this world in which we have very rich software services. Code that does something interesting, but also takes advantage of that connectivity for continuous delivery and updates, and delivered across multiple devices.

So that's the path we're on from an end user perspective.

Microsoft XML Commitment Leads The Industry
Pioneered the concept of XML Web services
Shepherded the standardization
Building broadest, deepest and earliest product support
Operate biggest XML Web services
Betting our business on XML Web services
Making customers successful today

Microsoft XML Commitment

Key concepts:

- Microsoft has led the XML efforts
- Our experience operating web services such as Passport gives us advantages in understanding how to create products for XML Web services
- Not just bolting onto new stuff – whole new generation of tools and infrastructure designed from the ground up to support XML model – VS.NET, .NET Framework, BizTalk Server

Script:

As I mentioned earlier, most of the rest of the rest of the industry has sort of jumped onto the XML Web services bandwagon. We think that's a great thing, we think that we have some unique advantages here. We really did pioneer this concept, a lot of this work was back in 1996, when there was this huge amount of frustration inside of Microsoft that HTML was a dumb terminal model. We wanted smart computers to talk to each other! And that drove XML, that drove SOAP, and these other standards, and really forced us to say, "Let's go back and rethink what software should look like in this world of ubiquitous connectivity."

We shepherded a lot of these things through standardization: we wrote or cowrote the various XML standards, SOAP started here, UDDI started here. From a product perspective, we think we have the broadest consistent and earliest product support. And our competitors are no espousing these ideas, they're on the right path, but they're just not going to get there as soon as we will.

We also operate these XML Web services in the real world. So things like Passport, being able to look at that backend from a product perspective is uniquely valuable.

XML Web Services Customers

XML Web Service Customers
Two weeks to integrate Windows NT-based reservation system with airline UNIX/CORBA solution; reusability unlocked additional business opportunities
Can bring new electronic payments banking partners online in as little as four hours
eBay commerce engine will be accessible from any Web site
Replacement of Web site with XML Web service for partner integration increased performance, reduced code by 80%
DOLLAR
checkspace
ebay
MSNBC

Key concepts:

- Simplicity of XML and related standards has made it possible for developers to create Web services solutions quickly and easily
- These solutions offer real business benefits, including:
    - Increased revenues
    - Decreased development costs
    - Decreased operational costs
    - Expanded customer base

Script:

There are a bunch of customers that are deploying this stuff today. Probably the most exciting thing about XML Web services is that it's kind of like the original Web in the sense that it's fairly easy for a small set of developers to take advantage of this stuff quickly and start to get real business benefit out of it. And that means that this will grow in a bottoms-up, organic way.

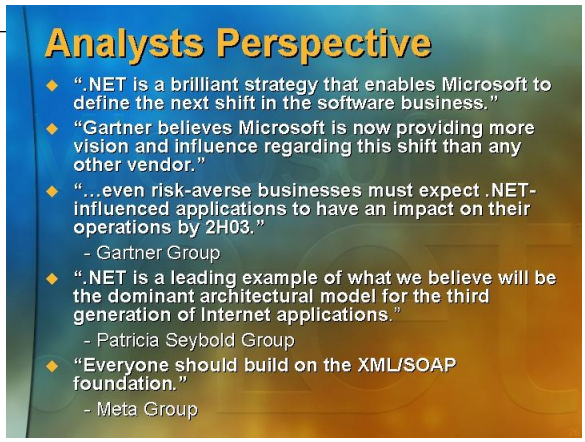
So companies like Dollar Rent-A-Car wanted to integrate their reservation system with Southwest Airlines' system. So you book a flight, and you get an offer: do you want to rent a car? Originally they got a quote from a traditional EAI mechanism that it was 6+ months, and that was the quoted schedule but in fact it was probably going to take a lot longer than that. But when they decided to do this integration between these heterogeneous systems using SOAP, it only took them about two weeks, and once they'd done that integration, they realized that they could bring any other airline they wanted on in a matter of days. It was reusable. By doing the booking directly, Dollar was able to significantly improve their bottom line for these reservations – they wouldn't have to pay the $8-9 fee that Galileo or SABRE would charge them to book through those intermediary systems. And in the first weekend it was live, without any promotion from either Southwest or Dollar, Dollar got 130,000 price quote requests from Southwest and booked an additional 2400 reservations.

CheckSpace is a company that does electronic payments. They can bring a new bank online in less than four hours. Banks are not renowned as the quickest and most flexible companies in the world, but this allows them to grow the network of people that they're providing payments across very very quickly.

eBay's not done yet, but they're taking their commerce engine, their auction engine, which is an interesting set of functionality, but probably more importantly it has an amazing amount of activity associated with that, and they're exposing that programmatically as XML Web services. That means that anybody that wants to embed auction capabilities that have both very very rich functionality but also deep liquidity can get that directly into their system.

MSNBC stopped doing random one-off integrations with all the different news feeds that they use and went to XML Web services, so they get reuse across their random feeds. They also found that their maintenance costs went down, and that the amount of code that they had to maintain went down by 80%, and they got better performance to boot.

Analysts Perspective


Analysts Perspective
".NET is a brilliant strategy that enables Microsoft to define the next shift in the software business."
"Gartner believes Microsoft is now providing more vision and influence regarding this shift than any other vendor."
"...even risk-averse businesses must expect .NET-influenced applications to have an impact on their operations by 2H03."
- Gartner Group
".NET is a leading example of what we believe will be the dominant architectural model for the third generation of Internet applications."
- Patricia Seybold Group
"Everyone should build on the XML/SOAP foundation."
- Meta Group

Key concepts:

- Analysts (including Gartner!) are forecasting that .NET will have significant impact on businesses
- .NET is a "brilliant strategy"

Script:

So the analysts are looking at what Microsoft is doing with XML Web services and .NET and are affirming the value of what we're doing. Even Gartner, who rarely has nice things to say about us, is acknowledging the impact of our vision for where computing is going.

Getting there: Developers

Getting There: Developers
Learn XML and related technologies
Work with Visual Studio.NET and .NET Framework betas
Start using Passport for user credentials
Keep your MSDN Universal subscription current

Key concepts:

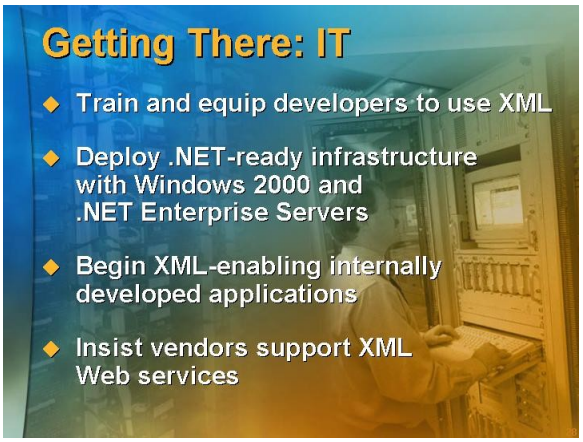
- Developers can get started now

Script:

In terms of getting there, if your developers don't have XML expertise, this is probably a good time to start building that expertise. We encourage you to take a look at Visual Studio.NET and the .NET Framework betas. Start thinking about services like Passport for managing user credentials, and you're going to have much more

flexibility in terms of make or buy, what things you build yourself and what things you use off the shelf. And as usual, the best way to keep up with this stuff is MSDN Universal.

Getting there: IT


Getting There: IT
Train and equip developers to use XML
Deploy .NET-ready infrastructure with Windows 2000 and .NET Enterprise Servers
Begin XML-enabling internally developed applications
Insist vendors support XML Web services

Key concepts:

- IT organizations can start now
- There are lots of other things IT folks should be doing, but which I'm not sure how to mention:
  - Preparing for a world in which much of their infrastructure is outside their walls
  - Exposing their value-add as XML Web services
  - Integrating legacy apps
  - Dealing with operational security issues
  - Passport?

Script:

IT organizations can get on the path for these changes by first equipping their developers to use XML – and insisting that vendors whose applications they use XML-enable those applications for simplicity of integration. They should also XML-enable internally developed and maintained applications. And the .NET Enterprise Servers and Windows 2000, because of their native support for XML and scale out technologies, represent a good foundation to start deploying.

Getting there: Business people

Getting There: Business People
Developer training
IT preparation
Reconsider business models and customer relationships
How does everything connecting to everything change my industry?

Key concepts:

- Business folks should prepare their developers and IT organizations, while considering the business implications of XML Web services on their business

Script:

Businesses who want to take advantage of XML Web services and .NET should get their developers trained, should get their IT staffs trained, but importantly, they should consider how integration can affect their business models. How can easy integration with partners help my business? How can easy integration with the applications that my customers use help my business? How can I provide new services or better customer service by integrating my applications? There is the potential for some fairly profound impact on most industries, and an opportunity for businesses to really revolutionize the way they reach customers and the way that they operate their businesses.

Summary

Summary
Fundamental shift: distributed computing for the Internet
XML Web services provide open, Internet-native integration mechanism
Microsoft .NET delivers most complete solution for XML Web services

Key concepts:

- We're at an industry inflection point around XML Web services
- XML Web services enable fast, easy integration
- .NET is the best, most complete platform for XML Web services

Script:

In summary, we're really excited about this new model for executing over the distributed Internet. Smart nodes talking to smart nodes, clients, servers, and services. XML Web services let us dramatically decrease the cost and the time and the energy and the difficulty of doing integration. And from a product standpoint, .NET is our platform for delivering on this XML Web services world, at the client level, at the services level, at the services level, and at the tools level. You can also expect Microsoft's higher-level, user facing applications like Office to embrace this model.

Wrapup

Key concepts:

- Open for questions. Common questions are around security, privacy, network bandwidth requirements, and business process orchestration, as well as

(in enterprise presentations) applications of this to the customer's specific situation or business.

Script:

That concludes my presentation. Questions?